

FAIRPLAY DRM和混淆实现

演讲者：

卢俊志 (@pwn0rz)、王馨笛 (@1dnix)、朱学文 (Ju Zhu)



DRM

Fairplay - DRM

介绍

数字版权保护

应用于电子书籍/音乐/视频

App DRM自2013年引入

私有代码，高度混淆

纵横

Fairplay - DRM

Load Command

```
$ otool -l target | grep -i crypt
    cmd LC_ENCRYPTION_INFO_64
        cryptoff 16384
        cryptsize 4177920
        cryptid 1
```

Fairplay - DRM

Fairplay Open - From Kernel



Fairplay - DRM

Fairplay Open - MIG

FairplayIOKit



fairplayd

```
#include <mach/std_types.defs>
#include <mach/mach_types.defs>

subsystem KernelUser unfreed 502;

type unk1_t = struct[136] of char;
type unk2_t = struct[84] of char;

routine fairplay_open(
    fairplay_port      : mach_port_t;
    executable_path   : pointer_t;
    cpu_type          : uint32_t;
    cpu_subtype       : uint32_t;
    out supf          : pointer_t;
    out unk_ool2      : pointer_t;
    out unk1          : unk1_t;
    out unk2          : unk2_t;
    out supf_size     : uint32_t;
    out ool2_size     : uint32_t;
    out ukn3          : uint32_t);
```

Fairplay - DRM

Fairplay Open - fairplayd

纵横

```
$ tree
.
├── SC_Info
│   └── target.sinf
└── target
    └── target.supf
```

Fairplay - DRM

Fairplay Open - SINF

纵横

```
$ sinf_view.py SC_Info/target.sinf
sinf.frma: game
sinf.schm: itun
sinf.schi.user: 0xdeadbeef
sinf.schi.key : 0x00000002
sinf.schi.iviv: <***16 bytes IV***>
sinf.schi.righ.veID: 0x00012345
sinf.schi.righ.plat: 0x00000000
sinf.schi.righ.aver: 0x11223344
sinf.schi.righ.tran: 0x11223344
sinf.schi.righ.sing: 0x00000000
sinf.schi.righ.song: 0x11223344
sinf.schi.righ.tool: P550
sinf.schi.righ.medi: 0x00000080
sinf.schi.righ.mode: 0x00000000
sinf.schi.righ.hi32: 0x00000002
sinf.schi.name:<***null terminated username, 256 bytes***>
sinf.schi.priv: <***432 bytes encrypted data***>
sinf.sign: <***128 bytes signature***>
```

Fairplay - DRM

Fairplay Open - SUPF

```
$ supf_view.py SC_Info/target.supf
KeyPair Segments:
    Segment 0x0: arm_v7, Keys: 0x3d0/4k, sha1sum = <code_sig>
    Segment 0x1: arm64, Keys: 0x3fc/4k, sha1sum = <code_sig>

Fairplay Certificate: <RSA 1024 Ceritificate, valid since 2008, expire at 2013>

RSA Signature: <128 bytes>
```

Fairplay - DRM

Fairplay Open - QA时间

1. 使用了不安全的RSA密钥长度, 没有校验RSA证书的有效期
3. SINF中明文存储了用户身份标识信息(但是沙盒内无法读取)
4. 可以通过调用MIG + Hook来稳定获取Fairplayd运行中间过程
5. 可通过回归测试确定最终和DRM相关/无关的字段
6. SINF文件中sinf.sign字段不校验 (仅在安装时通过installd校验)

Fairplay - DRM

Fairplay Decrypt - Kernel



Fairplay - DRM

Fairplay Decrypt - 一些细节

1. Fairplay 以page为单位解密，尺寸是4096 bytes
2. aes-128-cbc解密，密钥通过Fairplay Open的结果计算得出
3. 至少解密过程中没有涉及到HW AES(S8000)

Fairplay - DRM

Fairplay Decrypt - Demo

```
>>> calling Fairplay decrypt page : EQLZPp(handle=0xe421a923, off=0x36e000, src=0x101e6e000, dst=0x10080e800)
0x10e2f0 ==> IOMallocAligned(0x4000, 0x40) => 0x100808800
0xfea5c ==> IOMalloc(0x68) ==> 0x100204c80
0xfa7a4 ==> IOLockLock(0x1002061d0)
0xfb91c ==> IOUnlock(0x1002061d0)
0xfa7a4 ==> IOLockLock(0x100206220)
0x40924 ==> aes_decrypt_key(key=0x100206c82, len=0x10, ctx=0x16fdfff1d0)
0x4093c ==> aes_decrypt_cbc(in=0x101e6e000, iv=0x100206c92, n_blk=0x100, out=0x10080e800, ctx=0x16fdfff1d0)
aes-128-cbc key: 3ea81b294cf8107fc2646b7f6acde653, iv: 0888e773bb76ef698981d0733f1b2172
0xfb91c ==> IOUnlock(0x100206220)
0x10c094 ==> IOFree(0x100206c80, 0x68)
0x10e608 ==> IOFreeAligned(0x100808800, 0x4000)
>>> calling Fairplay decrypt page : EQLZPp(handle=0xe421a923, off=0x36f000, src=0x101e6f000, dst=0x10080e800)
0x10e2f0 ==> IOMallocAligned(0x4000, 0x40) => 0x100808800
0xfea5c ==> IOMalloc(0x68) ==> 0x100204c80
0xfa7a4 ==> IOLockLock(0x1002061d0)
0xfb91c ==> IOUnlock(0x1002061d0)
0xfa7a4 ==> IOLockLock(0x100206220)
0x40924 ==> aes_decrypt_key(key=0x100206c82, len=0x10, ctx=0x16fdfff1d0)
0x4093c ==> aes_decrypt_cbc(in=0x101e6f000, iv=0x100206c92, n_blk=0x100, out=0x10080e800, ctx=0x16fdfff1d0)
aes-128-cbc key: 668c99e6lac45e562986c89d350f2f67, iv: 97a8d8992c938907fffd89239c665fce
0xfb91c ==> IOUnlock(0x100206220)
0x10c094 ==> IOFree(0x100206c80, 0x68)
0x10e608 ==> IOFreeAligned(0x100808800, 0x4000)
Process _81249 exited with status = 0 (0x00000000)
```

混淆

Fairplay – 混淆

编译优化 vs makeOpaque

编译优化 : Constant Folding, Common Subexpression Elimination, Dead Code Elimination....

makeOpaque: 绕过编译优化

Expression* makeOpaque(Expression* in)



Fairplay – 混淆

makeOpaque: 不透明谓词

纵横

```
makeOpaque(true)  
=>  
uint32_t x = random();  
( (x * x % 4) == 0 || (x * x % 4) ==1)
```

Fairplay – 混淆

makeOpaque: 不透明谓词之
BogusCFG

```
if(makeOpaque(true)){
    real_block();
}else{
    fake_block();
}
```

Fairplay – 混淆

makeOpaque: 不透明常量之可逆变换

```
//对于互为模反元素的a: 4872655123和ra: 3980501275, 取
uint32_t x = random();
uint32_t c = 0xbeefbeef;
//则 -ra * c = 0x57f38dcb, 满足
((x * 4872655123) + 0xbeefbeef ) * 3980501275 + 0x57f38dcb == x
```

Fairplay – 混淆

makeOpaque:不透明常量之MBA表达式

```
//OperationSet(+, -, *, &, |, ~)
makeOpaque(x - c) => (x ^ ~c) + ((2 * x) & ~(2 * c + 1)) + 1;
```

Fairplay – 混淆

makeOpaque:不透明常量应用-
IndirectBranch

```
//OperationSet(+, - , * , & , | , ~)
jmp branch;
=>
jmp global_branch_lut[index];
=>
jmp global_branch_lut[makeOpauqe(index)];
```

Fairplay – 混淆

静态恢复实战 – Call Graph 恢复

Indirect Branch + Call Convention
 混淆的同时对参数进行了混淆（父函数加密，子函数解密，利用 LLVM 不进行 Inter-procedure 分析的特性）

```

__text:0000000000043888 E1 BF 00 F9      STR      X1, [SP,#8x98+var_78]
__text:000000000004388C A8 63 00 D1      SUB      X8, X29, #-var_18
__text:0000000000043890 09 0D 00 52+     MOV      W9, #0x36724068
__text:0000000000043890 49 CE A6 72
__text:0000000000043898 08 01 00 4A      EOR      W8, W8, W9
__text:000000000004389C A9 6C 06 52+     MOV      W9, #0x1AA53365
__text:000000000004389C A9 54 A3 72
__text:00000000000438A4 08 7D 00 1B      MUL      W8, W8, W9
__text:00000000000438A8 09 31 96 52+     MOV      W9, #0xD9548188
__text:00000000000438A8 89 2A 00 72
__text:00000000000438B0 08 01 00 4A      EOR      W8, W8, W9
__text:00000000000438B4 A8 03 1F 88      STUR     W8, [X29,#var_18]
__text:00000000000438B8 E8 23 00 91      ADD      X8, SP, #0x90+var_88
__text:00000000000438BC A8 83 1E F8      STUR     X8, [X29,#var_18]
__text:00000000000438C0 68 07 00 80+     ADRL     X8, off_1303FB
__text:00000000000438C0 68 C1 0F 91
__text:00000000000438C8 09 01 48 F9      LDR      X9, [X8] ; _fp_dh_d2422d6a5f479457076ea15ed54da5a1
__text:00000000000438CC A8 3F F6 F2      MOVK    X8, #0xB1FD,LSL#48
__text:00000000000438D0 A0 63 00 D1      SUB      X0, X29, #-var_18
__text:00000000000438D4 28 09 3F D7      BLRAA   X9, X8
__text:00000000000438D8 A0 43 5F 88      LDUR    W8, [X29,#var_C]
__text:00000000000438DC A8 83 5F F8      LDUR    X8, [X29,#var_B]
__text:00000000000438E0 29 07 00 80      ADRP    X9, #off_128168@PAGE
__text:00000000000438E4 29 B5 40 F9      LDR     X9, [X9,#off_128168@PAGEOFF]
__text:00000000000438E8 29 01 40 F9      LDR     X9, [X9]
__text:00000000000438EC 3F 01 00 EB      CMP     X9, X8
__text:00000000000438F0 81 00 00 54      B.NE    loc_43C00
__text:00000000000438F4 FD 7B 49 A9      LDP     X29, X30, [SP,#0x90+var_s8]
__text:00000000000438F8 FF 83 02 91      ADD     SP, SP, #0xA8

```

irplay – 混淆

态恢复实战 – Call Graph 恢复

数混淆恰巧在父子
数中引入**相同随机**
，让我们得以根据
一特性恢复出调用
系

```

--text:0000000000043888 E1 8F 00 F9      STR      X1, [SP,#0x90+var_78]
--text:000000000004388C A8 63 00 D1      SUB      X8, X29, #-var_18
--text:0000000000043890 09 0D 00 52+      MOV      W9, #0x36724068
--text:0000000000043890 49 CE A6 72      ADD      W8, W8, W9
--text:0000000000043890 00 01 00 0A      EOR      W9, #0x1AA53365
FF 23 03 05          STP      X28, X27, [SP,#0x90+var_58]
FA 67 B4 A9          STP      X26, X25, [SP,#0x90+var_50]
FB 5F B2 A9          STP      X24, X23, [SP,#0x90+var_50]
F6 57 03 A9          STP      X22, X21, [SP,#0x90+var_28]
F4 4F 00 A9          STP      X20, X19, [SP,#0x90+var_18]
FD 7B 05 A9          STP      X19, X18, [SP,#0x90+var_18]
FD 43 B1 91          ADD      X19, SP, #0x58
FF 03 07 D1          SUB      SP, SP, #0x100
F3 03 00 A8          MOV      X19, X8
FB 63 02 91          ADD      X24, SP, #0x210+var_178
E8 03 00 98          ADRP   XB, #off_128168@PAGE
E8 03 00 98          LDR    XB, [XB,#off_128168@PAGEOFF]
E8 01 00 F9          LDR    XB, [XB]
A8 03 1A F8          STUR   XB, [X29,#var_48]
E8 0C 00 52+          MOV    W25, #0x1AA53365
E8 54 A3 T2          MOV    W23, #0x25409993
77 32 00 52+          MOV    W8, #0x36724068
77 05 A6 T2          LDR    X9, [X8] ; _fp_dh_d2422d6a5f479457076ea15ed54da5a1
E8 00 00 52+          MOVK   X8, #0xB1FD,LSL#48
E8 CE A6 T2          SUB    X8, X29, #-var_18
E8 02 00 54          EOR    W8, W19, W8
E8 70 19 58          LDR    W8, W8, W25
E8 00 00 59          LDR    W9, [XB,W8]
E8 01 00 54          EOR    W11, W9, W8
E8 00 00 F9          LDR    X13, [XB]
A8 19 00 59          LDR    XB, [X13,#0x30]
E8 00 00 F9          LDR    X20, [X13,#0x18]
E8 01 00 F9          LDR    XB, [X13]
E8 01 17 00          ADD    W21, W11, W23
AA 3A 29 51          SUB    W10, W21, #0x4E
DC 00 00 59+          ADRL   X12, _fp_dh_15d3f11e9574fbad3d4286ee9e200ff924
E8 01 12 91          CMP    X9, X8
E8 09 00 58          LDR    X10, [X12,W10,SLT#0]
E8 39 00 D1          SUB    X22, X18, #0x4E
E8 19 00 52          MOV    W10, #0x58C8
E8 09 1F 07          BRAA  XB, X18

```

Fairplay – 混淆

静态恢复实战 – 尝试恢复CFG

1. 使用了Indirect Branch混淆机制
2. 同一个函数的每个基本快具有相同的PAC Modifier
3. 全局跳转表中DYLD Chained Fixup中含有Modifier信息
4. 但基本快之前目前仍然是孤立的，需要动态恢复

```
loc_FE434
MOV W9, #0x43 ; 'C'
MADD W9, W8, W9, W13
ADD W9, W9, W14
LDR X9, [X12,W9,SXTW#3]
MOV W10, #0x44DE7E2F
ADD W25, W10, #0x16
MOV W10, #0x18A3
BRAA X9, X10

loc_FE458
W20, W8, W13, W14
MADD W8, W20, #0xA8
SUB W8, W20, #0xA8
ADD X8, X26, W8,SXTW#3
LDR X9, [X8]
MOV W10, #0x407B
ADD X10, X10, W20,UXTW
BFI X8, X10, #0x30, #0x10 ; 'B'
MOV W9, #0x68 ; 'h'
MOV X19, X4
BLRAA X9, X8
LDUR X13, [X29,#var_90]
LDUR X12, [X29,#var_60]
STR X9, [X28,#0x178]
CMP X9, #0
CSET W8, EQ
CSET W9, NE
MOV W10, #0xC3
MADD W10, W9, W10, W20
LDR X10, [X12,W10,SXTW#3]
MOV W11, #0x18A3
MOV W25, W8,44DE7E2F
BRAA X10, X11
```

Fairplay – 混淆

静态恢复实战 – 尝试恢复CFG

1. 使用了Indirect Branch混淆机制
2. 同一个函数的每个基本快具有相同的PAC Modifier
3. 全局跳转表中DYLD Chained Fixup中含有Modifier信息
4. 但基本快之前目前仍然是孤立的，需要动态恢复

```
loc_FE434
MOV W9, #0x43 ; 'C'
MADD W9, W8, W9, W13
ADD W9, W9, W14
LDR X9, [X12,W9,SXTW#3]
MOV W10, #0x44DE7E2F
ADD W25, W10, #0x16
MOV W10, #0x18A3
BRAA X9, X10

loc_FE458
W20, W8, W13, W14
MADD W8, W20, #0xA8
SUB W8, W20, #0xA8
ADD X8, X26, W8,SXTW#3
LDR X9, [X8]
MOV W10, #0x407B
ADD X10, X10, W20,UXTW
BFI X8, X10, #0x30, #0x10 ; 'B'
MOV W9, #0x68 ; 'h'
MOV X19, X4
BLRAA X9, X8
LDUR X13, [X29,#var_90]
LDUR X12, [X29,#var_60]
STR X9, [X28,#0x178]
CMP X9, #0
CSET W8, EQ
CSET W9, NE
MOV W10, #0xC3
MADD W10, W9, W10, W20
LDR X10, [X12,W10,SXTW#3]
MOV W11, #0x18A3
MOV W25, W8,44DE7E2F
BRAA X10, X11
```

Fairplay – 混淆

静态恢复实战 – 其他未解决的

1. 基于不透明常量的数据流混淆, 目前未找到其生成规则

```
v220 = v22;
v40 = *(&fp_dh_5d6e44dfcb68b4b624f58fc587dbb98d + v25 + 38) - 6;
v41 = *(&fp_dh_5d6e44dfcb68b4b624f58fc587dbb98d + v25 + 104) - 10;
v230 = (*&v41[4 * (v34 ^ 0xE4)] - 2104345745) * *&v40[4 * (v24 ^ 0xA4)] * (v39 - ((2 * v39) & 0xF056D28A) + 21251);
v221 = v22;
v216 = v226 * v230 * 0x654F64EA * 0x33A1F418;
v224 = v22 * v216 * 0x72C8BD2A;
v42 = v224 * v224 * 0x134F22CB;
v43 = (((v225 * v224) * 0xC49CAE1) - ((2 * ((v225 * v224) * 0xC49CAE1)) & 0xEC | 0x82988800) + 360154486) * *&v3;
v44 = (v43 - ((2 * v43) & 0xF056D28A) + 2125162757) * *&v40[4 * (BYTE2(v42) ^ 0xE5)];
v223 = (*&v41[4 * (BYTE1(v42) ^ 0x21)] - 2104345745) * *&v38[4 * (HIBYTE(v42) ^ 0x39)] * v230 * 0x654F64EA * ((H));
v45 = v22 * v223 * 0x3F98D223;
v222 = v45 * v42;
v46 = v45;
v218 = v45;
v47 = v45 ^ v42 ^ 0x94592E63;
v48 = *&v41[4 * (((v45 ^ v42 ^ 0x2E63) >> 8) ^ 0xDC)];
v49 = (v48 - ((2 * v48 + 86275886) & 0x7F31A446) - 1037365870) * *&v38[4 * (HIBYTE(v47) ^ 0xCB)] * v223 * 0x3F98D223;
v215 = v216 * v223 * 0x3F98D223;
v214 = *&v37[4 * (v47 ^ 0xEF)] * *&v40[4 * (BYTE2(v47) ^ 0xFE)] * ((v47 ^ 0xC49CAEB2)
- ((2 * (v47 ^ 0xC49CAE02)) & 0xEC | 0x82988800)
+ 360154486) * (v49
- ((2 * v49) & 0xF056D28A)
+ 2125162757);
v50 = v215 * v214 * 0x258D9763;
```

Fairplay – 混淆

动态调试工具-穷人的“内核驱动”调试器

1. 把FairplayIOKit内核驱动加载到用户态
2. 通过dyld的机制通知调试器新加载的内核扩展
3. 开始调试

```
0xfca0c => host_priv_self() => 0xc03
0xfca38 => host_get_special_port(priv=0xc03, node=-1, which=17, port=0x100808a10[0,
0xff1e8 => strlen("/Applications/ ■ ■ ■ /WrappedBundle/ ■ ■ ") => 0x31
0xff2b4 => kmem_alloc(map=0xa858000080000000, addrp=0x100809000, size=32, tag=0x100
0xff3b8 => vm_map_copyin(src_map=0xa858000080000000, src=0x100206930, len=32, src_c
Process 81249 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 2.1 3.1
  frame #0: 0x0000000100627adc FairPlayIOKit`uf_setup_from_fp
FairPlayIOKit`uf_setup_from_fp:
→ 0x100627adc <+0>: pacibsp
  0x100627ae0 <+4>: sub    sp, sp, #0x180          ; =0x180
  0x100627ae4 <+8>: stp    x24, x23, [sp, #0x140]
  0x100627ae8 <+12>: stp    x22, x21, [sp, #0x150]
Target 0: (uploader) stopped.
```

Fairplay – 混淆

动态调试工具-执行流跟踪

可以记录自己的执行路径(trapfuzz类似)

可以记录很多次非直接跳转的结果(trapfuzz不支持)

不能single-step自身

从DTrace中获取灵感: exception-emulation-recover

```
static void breakpoint_handler(int signum, siginfo_t* info, void *context){  
    // get breakpoint address and context  
    uint32_t *pc = (uint32_t*)info->si_addr;  
    ucontext_t *ctx = (ucontext_t*)context;  
  
    uint32_t *saddr = shadow_addr(pc);  
    uint32_t opcode = *saddr;  
  
    uint64_t rn = 0;  
    uint64_t ctx_data = 0;  
    uint64_t ncov = 0;  
  
    //now let's interpret arm64 instructions lol  
    if((opcode & 0xfffffc00) == 0xd73f0000){ /*blraa*/  
        // lr = pc + 4  
        // br rn  
  
        rn = (opcode >> 5) & 0x1f;  
        ctx_data = ctx->uc_mcontext->_ss._x[rn]; //blr target  
  
        ctx->uc_mcontext->_ss._lr = (uintptr_t)pc + 4;  
        ctx->uc_mcontext->_ss._pc = ctx_data;
```

Fairplay – 混淆

动态调试工具-Demo

```
0xd2fd8: cset true
0xd2fdc: cset false
0xd3004: b 0xd3008
0xd3014: cset true
0xd3018: cset false
0xd3040: b 0xd3048
0xd3080: bl 0x3d4f0
0x3d504 => memcpy(dst=0x11c80a540,src=0x120008010,size=0x10)
0xd30a8: b 0xd3114
0xd3128: cset false
0xd3130: cset false
0xd313c: cset false
0xd3180: b 0xd3184
0xd31d0: bl 0x3d4f0
0x3d504 => memcpy(dst=0x11c80a090,src=0x120008000,size=0x20)
```

Fairplay – 混淆

动态调试工具-更多可能

WIP 反射式macho注入

WIP 无源代码的macho二进制Profiler工具

Update @ https://github.com/pwn0rz/fairplay_research

感谢观看！

KCon 汇聚黑客的智慧

